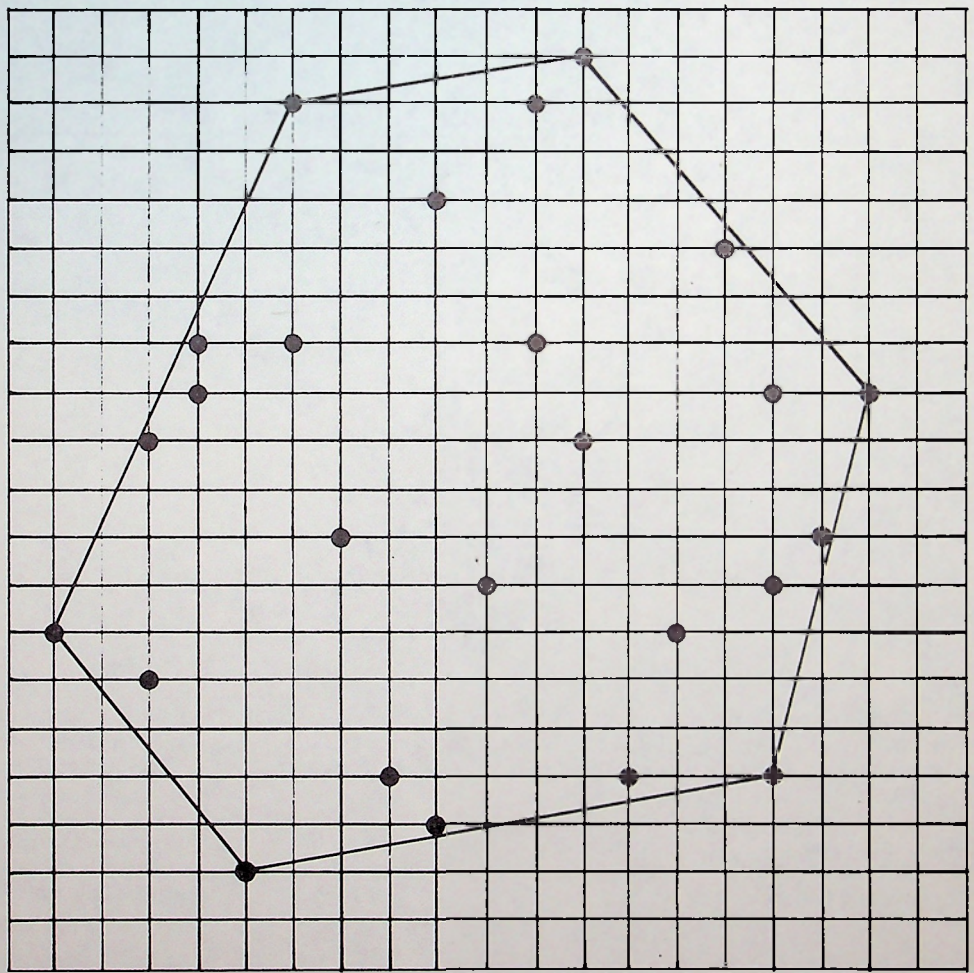
$2.50

48

Popular Computing

March 1977 Volume 5 Number 3



THE RUBBER BAND PROBLEM

A group of points is located in the square region between zero and 20 on both axes. Their coordinates are given as input data.

A program is to be written to print a list of those points which, when connected by straight lines, will form a convex polygon that will just enclose all the points. In simple terms, we want "to put a rubber band around all the points," as Clarence Schmidt has put it.

In the example shown on the cover, the input to the program would be these coordinates:

5,2	3,11	12,11	9,16
3,6	1,7	8,4	13,4
16,4	10,8	16,8	4,12
6,13	11,13	16,12	14,7
17,9	18,12	6,18	11,18
12,19	15,15	4,13	9,3
7,9			

and the output should be:

1,7 1,18 12,19 18,12 16,4 5,2

in that cyclic order. □

PROBLEM 163

POPULAR COMPUTING is published monthly at Box 272, Calabasas, California 91302. Subscription rate in th United States is $20.50 per year, or $17.50 if remittance accompanies the order. For Canada and Mexico, add $1.50 pe year to the above rates. For all other countries, add $3.50 per year to the above rates. Back issues $2.50 each. Copy right 1977 by POPULAR COMPUTING.

Editor: Audrey Gruenberger
Publisher: Fred Gruenberger
Associate Editors: David Babcock
Irwin Greenwald

Contributing editors: Richard Andree
William C. McGee
Thomas R. Parkin

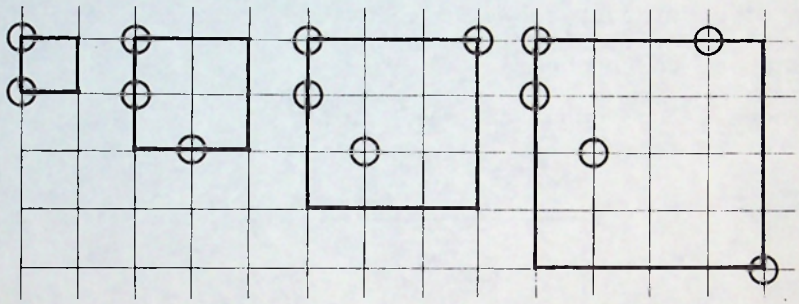
Advertising Manager: Ken W. Sim
Art Director: John G. Scott
Business Manager: Ben Moore

N SPOTS

The following problem appeared in the January 1972 Journal of Recreational Mathematics, and is due to Sidney Kravitz:

> A square lattice has N points on one side. We wish to place N spots on the lattice points so that all the distances between pairs of spots are distinct.

Solutions for cases N = 2, 3, 4, and 5 are shown here, with all (squared) distances listed:

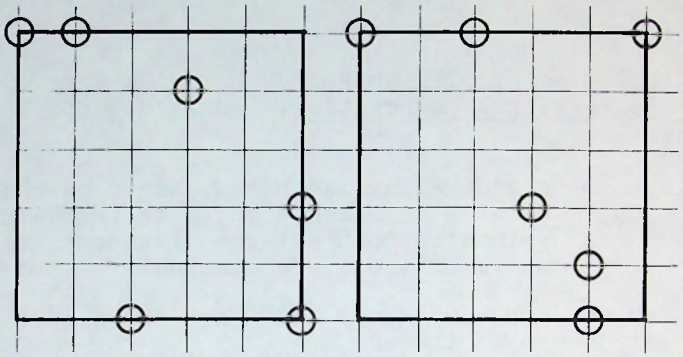


1	1,2,5	1,2,5, 8,9,10	1,2,5,8,9,10 13,25,32,26

The N = 6 case is the first difficult one, because of the possibility of having an integral distance of 5 in two different ways.

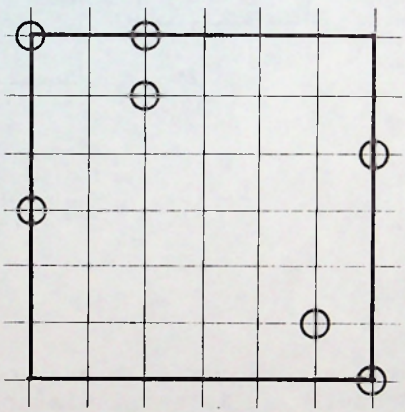
Associate Editor David Babcock programmed the general case, using the backtracking technique (see the articles by Thomas R. Parkin in issues 33 and 34 for a discussion of backtracking). For each case, starting with N = 6, the program essentially tests every way in which N things can be distributed among N^2 positions (that is, $_{n^2}C_n$). This number grows rapidly with N, as Table A shows. The fourth column of Table A gives the actual number of configurations for which the distances were calculated.

There are two unique solutions for N = 6:



The computer printout shows 16 solutions; that is, each of the unique solutions is given in the 8 possible ways that it can appear on the board.

There is only one solution for N = 7:



Most important, there is no solution for N = 8. In the case N = 6, there are 19 possible distances on the 6 x 6 lattice. Six points have 15 distances between them. Therefore, if the 15 distances are to be distinct, 4 of the possible 19 distances will be omitted. For the two solutions, the (squared) distances that are omitted are:

2, 16, 18, and 32

8, 16, 34, and 50.

The September 1976 issue of the Journal discusses the problem at length, showing all the results given here plus proofs that no solution is possible for N greater than 14, and computer evidence that there is no solution for N = 9. Thus, cases for N = 10, 11, 12, 13, and 14 are still open for investigation.

Similarly for N = 7; there are 25 possible distances and the solution, using 21 of them, omits:

18, 26, 32, and 61.

For N = 8, there are 41 possible distances, of which 28 are to be used. This would appear to make case 8 more tractable, except that now, for the first time, this complexity is introduced:

distance 50 is possible by 5 and 5 or by 1 and 7
distance 65 is possible by 1 and 8 or by 4 and 7
distance 25 is possible by 5 and 5 or by 3 and 4,

which may explain why no solution is possible. For N = 9, there are 49 distinct distances possible, of which only 36 are needed, so the question of whether or not a solution for that case is possible is still open.

Case	Number of possible configurations	Machine time (seconds)	Actual number of configurations tested by backtracking
2	6		
3	84	.5	
4	1820	3	1744
5	53130	10	20912
6	1947792	50	194656
7	85900584	450	1644839
8	4426165368	3600	14080163
9	260887834350		
10	17310309456440		
11	1.276749965 E15		
12	1.036192938 E17		
13	9.176358301 E18		
14	8.805305164 E20		

Table A.

CONTEST 12 RESULTS

Our 12th Contest (COUNTDOWN) appeared in issue No. 43. We start with the 150th term of the Fibonacci sequence:

9969216677189303386214405760200

and subtract some constants, the first of which is 29!:

8841761993739701954543616000000

then 10^{19} times 22!:

1124000727777607680000000000000

and so on. The most difficult of the terms to calculate was the ninth, which was

the integral part of 10^{13} times the 7th root of 306:

22651817862956.

The end result of all the arithmetic involved was the number 272, which is our postal box number. The $25 prize goes to Jeff Shallit, Wynnewood, Pennsylvania. □

■■■■■□□□□□

Log 48	1.68124123737558721814998348215308741627288839003913 0
ln 48	3.87120101090789092906417372275523197694949109526377 0
$\sqrt{48}$	6.92820323027550917410978536602348946777122101524152 3
$\sqrt[3]{48}$	3.63424118566427931778242351265452100485642092628243 9
$\sqrt[10]{48}$	1.47273335753468870339620067262550950956588373816235 8
$\sqrt[100]{48}$	1.03947108337618388328266547290980149324210861193075 9
e^{48}	701673591209763173865.471599886117405455937998725321 9 837545488788484222472082619783 3
π^{48}	729783246826064035206280.518927674569021545935467395 7 239829840096691073598760736 8
$\tan^{-1}$ 48	1.54996600675867953427823581338689203100433752475424 8

N-SERIES 48

Cycle Stealing

Quite frequently, the computer printouts that arrive, bearing solutions to one of POPULAR COMPUTING's problems, also bear clear indications that some benevolent corporation has unknowingly contributed computer time to work that can be no part of the corporation's business.

Occasionally one sees an acknowledgement of this fact of life in the literature, when a footnote to some scholarly work says "The author wishes to express his gratitude to the Dirigible Corporation for the use of their computer"--this being the first time that the Dirigible works knows about it.

Technically, using one's employer's computer for personal work is embezzlement, no matter how often the user mutters "The machine would have been idle otherwise." However, it is almost unheard of for employers to complain about such practices; perhaps they regard it as a necessary fringe benefit for their sensitive programmers, or as a beneficial contribution to their employee's education.

Donn Parker doesn't agree. In the article "Donn Parker on Computer Abuse" in the first issue of Personal Computing, he says:

> "Programmers tend to think of computer time as their domain. They have a "right" to computer time because of their technology. It's easy for them to rationalize: "There are the machine cycles not being used. Why in the world don't I use them for some purpose?"
>
> "I've run into that as manager of a data center. People want to use the machine at night to run roulette odds, that sort of thing, and if they ask me, I say no. We can't do that. We can't let somebody use at no cost what we charge other people for."

The rub is, of course, that programmers don't usually ask. It's in the nature of programming work that anyone can make almost unlimited computer runs without anyone knowing just what is being run.

Mike Beeler, Cambridge, Massachusetts, has invented a compromise solution to this problem, to insure at least that the personal job really operates at the noise level of the system.

> "I was running within an operating system which does not provide (so far as my friends and I could figure out) any way to run a job at 'priority epsilon.' That is, I wanted to have my program run only when there were no 'legitimate users' competing for time. The system does have capability for batch submissions, and for delayed activation (to start up a job in the middle of the night, for instance), but in both cases the job, once begun, will compete on an equal basis against legitimate jobs.
>
> "So, what I did was to write a little supervisor program. Every minute it wakes up and gets the one-minute load average from the operating system. If this is below .6 and the supervised program is not running, the supervisor resumes it. If the load average is above 1.6 and the supervised program is running, the supervisor suspends it. Otherwise it does nothing, and goes back to sleep for another minute. It worked very well, letting me rack up several hours of machine time during nights and never annoying anyone."

Mr. Beeler also makes these points:

1. Use of the company's unused cycles is sometimes acknowledged explicitly as a fringe benefit.

2. "Hacking" can uncover hardware and software bugs that might not otherwise be revealed. A benefit of having someone testing the computer in ways not often exercised is increased reliability to the legitimate users.

3. Some consideration should be given to the "worth" of the program. I freely admit that this is a very vague area, but "common sense" can distinguish some uses of the free cycles from others.

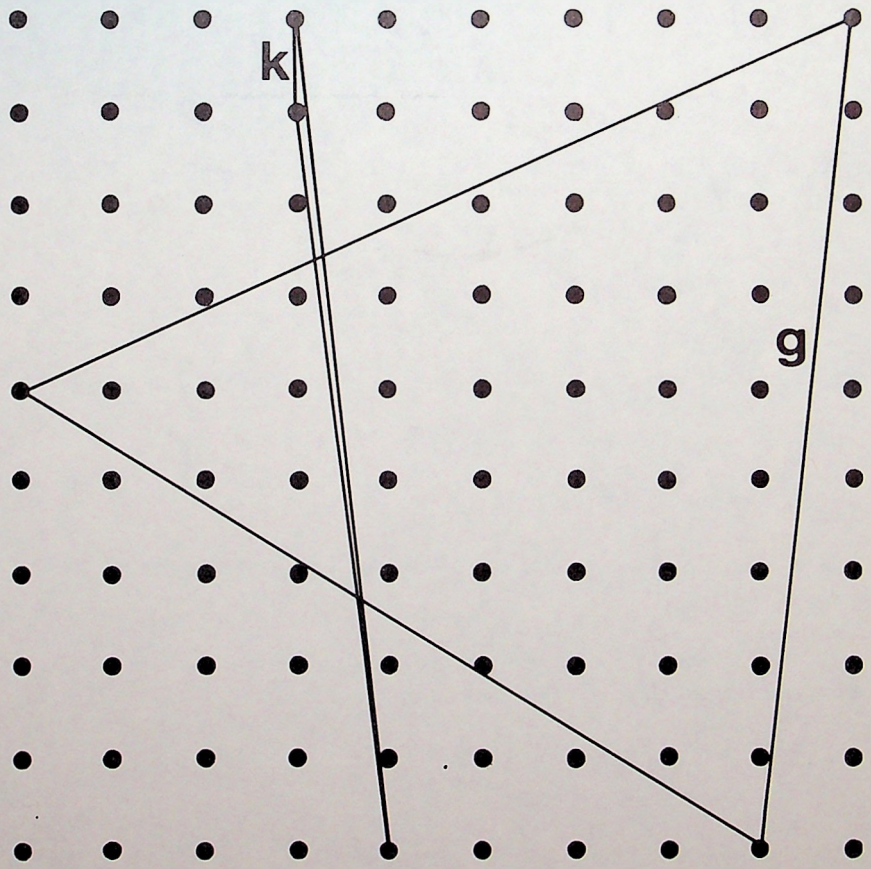
4. Some operating systems explicitly include mechanisms to make some users "more equal" than others ("equal" in a joking sense). For instance, the MIT Artificial Intelligence Lab's ITS (Incompatible Timesharing System) contains a mechanism to give any specific user the CPU time which 2**n users would get. n defaults to 0, so changing your n to 1 gets you the time that two users would get on a system with one more user than before incrementing your n; and so forth. Since n can be negative, you can make it, say, -6, which will interfere negligibly with other users so long as there <u>are</u> other users, but then will give you all the time when there are none. This, for instance, is superior to my supervisor program in that any number of "illegitimate" users can be competing gracefully with each other. Two users using my supervisor might or night not get equivalent performance; timing considerations would dictate.

THOSE 157252 TRIANGLES AGAIN

On a grid of 10 x 10 points, there are 157252 possible real triangles (that is, with non-zero area) made by connecting three points at a time.

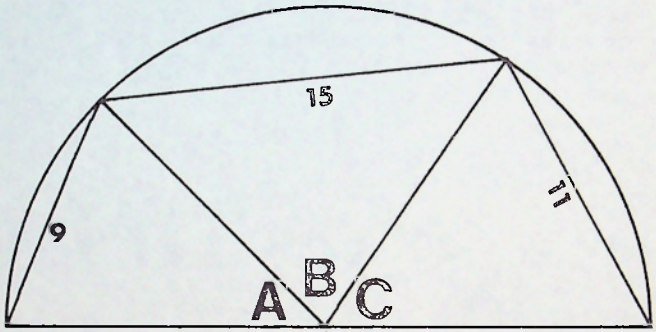
For a triangle like (k) in the figure, the ratio of the perimeter to the area is 36.235286. For triangle (g), on the other hand, that ratio is .73605777.

Are those the extremes? Are there triangles with coordinates on the 10 x 10 grid for which the ratio of perimeter to area is outside that range? How many different ratios are possible, and how many triangles are there of each possible ratio?



THREE CHORDS

In the September issue of 65 Notes (Vol. 3, No. 8), Terence O'Neill uses an old puzzle problem as an example of the use of approximation techniques.



Given the three chords inscribed in a semi-circle, find the radius. We have:

$$A + B + C = 180°$$

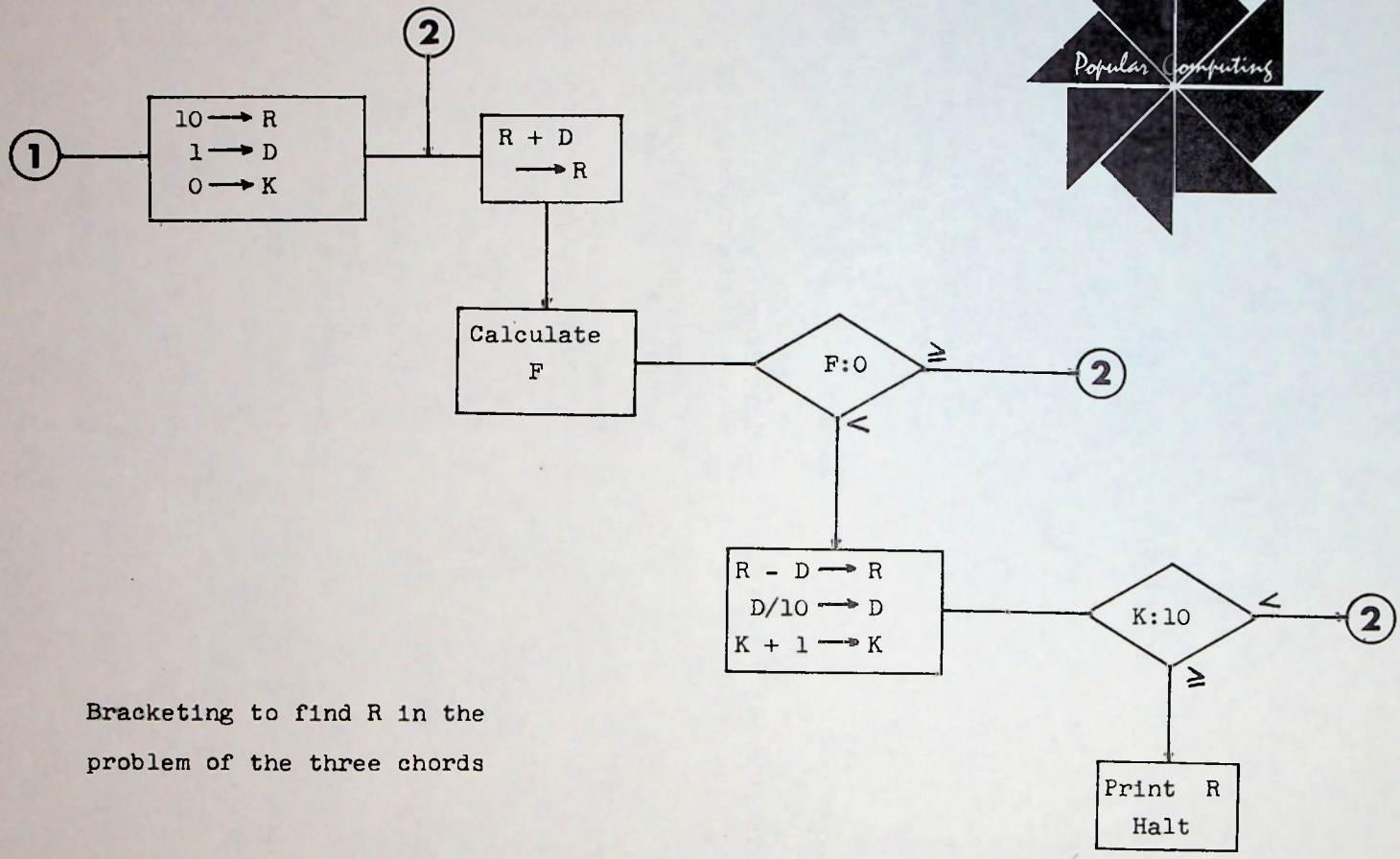
$$\frac{4.5}{R} = \sin\frac{A}{2}$$

$$A = 2 \sin^{-1} \frac{4.5}{R}$$

which leads to the equation:

$$F = \sin^{-1}\frac{4.5}{R} + \sin^{-1}\frac{7.5}{R} + \sin^{-1}\frac{5.5}{R} - 90 = 0$$

and R can readily be found by bracketing, as shown in the accompanying flowchart.



Bracketing to find R in the problem of the three chords

POPULAR COMPUTING is still the only magazine devoted exclusively to computing, as an art, a recreational activity, and a serious subject. It fosters the theme that

THE WAY TO LEARN COMPUTING IS TO COMPUTE

and to that end presents many problems for computer solution (most of them not solvable without computers). Much of its contents is designed for teachers of computing, to liven up classroom work. All of the material is original and unobtainable elsewhere. The essays on The Art of Computing are unique and form a significant contribution to computing literature, as do contributed articles by outstanding computing experts.

Listed here, in no special order, are other magazines for computer users, pocket calculator users, and computer hobbyists.

INTERFACE AGE	6615 Sunset Boulevard Hollywood, California 90028 (a committee of editors) Monthly, $10 per year.
BYTE	70 Main Street Peterborough, New Hampshire 03458 Carl Helmers, Jr., editor Monthly, $12 per year.
CREATIVE COMPUTING	Box 789-M Morristown, New Jersey 07960 David Ahl, editor Bi-monthly, $15 per year.
KILOBAUD	73 Magazine Street Peterborough, New Hampshire 03458 Monthly, $15 per year.
PERSONAL COMPUTING	167 Corey Road Brookline, Massachusetts 02146 Nels Winkless, III, editor Monthly, $8 per year.

MITS COMPUTER NOTES	2450 Alamo, S. E. Albuquerque, New Mexico 87106 Andrea Lewis, editor Monthly, house organ
65 NOTES	2541 W. Camden Place Santa Ana, California 92704 Richard J. Nelson, editor Monthly, $12 per year.
52 NOTES	9459 Taylorsville Road Dayton, Ohio 45424 Richard C. Vanderburgh, editor Monthly, $12 per year.
PEOPLE'S COMPUTER COMPANY	Box 310 Menlo Park, California 94025 Bob Albrecht, editor Bi-monthly, $5 per year.
dr. dobb's journal of Computer Calisthenics & Orthodontia	Box 310 Menlo Park, California 94025 Jim Warren, Jr., editor Monthly, $10 per year.

65 Notes is for users of the Hewlett-Packard line of programmable pocket calculators; 52 Notes for the users of the Texas Instruments machines. Byte, Interface Age, Kilobaud, and the MITS Computer Notes are aimed at the hobbyists who build their own computers. Creative Computing and the two publications of Dymax, Menlo Park, deal in the recreational aspects of computer use; all three specialize in BASIC programs. Personal Computing is new and will have to find its character and its niche. The premier issue contains: an interview with Donn Parker on computer crime; "Ten Easy Steps to Becoming a Computer Hobbyist," "Hard Talk About Hardware," "An Informal History of the Computer Market," "Let's Improve BASIC," predictions on Future Computing by Dick Heiser, and other articles.

Schwartz on CALCULATORS

Where is the dividing line between calculators and computers? That question has been with us a long time, but has recently become controversial again with the emergence of the powerful programmable machines (the SR-52, SR-56, HP-65, HP-67, HP-97, and the new National NS-7100).

In May of last year, Dr. Mordecai Schwartz, writing in 65-Notes, referred to the HP-65 as being definitely a computer. Dr. Schwartz is a prolific writer and, when challenged, brought forth not only a discussion of the calculator vs. computer question, but also much insight into the power and limitations of the new machines.

Gruenberger's views on the former subject were included in the book review that appeared in issue No. 46, page 6. Dr. Schwartz's views merit attention. He owns both an SR-52 and an HP-65 (and thus is above the largely-emotional debates over algebraic vs. RPN notation). He has written numerous ingenious and tight programs for the libraries of both machines (including the fascinating "Sequences" program--see issue No. 42, page 5). Following are excerpts from numerous letters of Dr. Schwartz's:

Thanks for taking the time to review with me your thoughts on the use of the term "computer." Obviously, I didn't realize how firmly embedded the program-modifiability criterion is to the currently accepted definition, or your contribution to that acceptance.

The just-announced HP-67 will then, despite its order-of-magnitude increase in power and capability, be no more a computer than the HP-65, and this despite its 224 program steps, fully merged codes, indirect addressability, "smart card" reader, separate or merged program and data read-in, register manipulation capabilities, expanded flag functions and subroutine levels, etc. I fully anticipate that it will significantly outperform the SR-52, yet by the program-modifiability criterion has less claim to the term "computer."

It seems very hard to call the HP-67 a "calculator" ...yet the HP-67 is sufficiently superior to the HP-65, both quantitatively and qualitatively, that it will open up whole new programming areas for the hand held machines.

I have a feeling that your sharp distinctions may become somewhat blurred if manufacturers such as Hewlett-Packard continue to expand calculator capability in current directions without making the stored program modifiable. After all, to elaborate upon your example, a program step like STO 12 can readily be altered to STO 13 by a change in the i-register index on the HP-67 if the actual step is STO(i). The same program code sequence, then, may execute differently on different passes.

I understand and appreciate the need, both at a theoretical and a practical level, for exactitude in the concept and definition of "computer." If I had to pick one criterion, I certainly couldn't do better than to accept the concept of full program self-modifiability as the dividing line between calculator and computer.

However, let me be the devil's advocate for one more go-around. You felt that "changing operations from + to - can't be faked by indirect addressing; it requires true address modification." Actually, this alteration is just as easy, on a machine like the HP-67, as changing STO 12 to STO 13. At a step where you anticipate you might need ⊕ on one occasion and ⊖ on another, simply insert SBR(i).

SBR(i) tells the program to jump to the program step stored in the I (index) register, execute the subroutine, and return to the 'main sequence' step that follows SBR(i). Let the subroutine at program location i_1 be ⊕ and at i_2 ⊖; then when $i = i_1$, plus is executed and when $i = i_2$ minus is executed. The decision as to choice of operation is made equivalent to the choice of one integer rather than another by intermediate results at distant points.

Obviously this process is quite general. At <u>any</u> desired step <u>any</u> available computer operation (or NOP) may be performed. At the desired program step, again use SBR(i) where i = 0, 2, 4, 6,... for n values, where n is the number of callable computer functions, digits, etc. Assuming each computer function is callable in a single program step (if not, the i values are just spaced further apart), we need only 2 program steps for each such function--one for the function itself and one for RTN.

Thus, each callable computer operation is mirror-imaged into an indirectly addressable equivalent function, with all n functions occupying 2n steps of program space. Any of these functions can be interchanged at any program step--if that step is SBR(i)--by an I address change (choice of integer) at some distant program step. An I-list is needed to change several steps at one time.

The process can be generalized in still other directions. For example, with a second index register J (or without it, via an I-register pyramiding process), any (i,j) indexed pair of integers will not only specify the desired "mirror image" function via i, but also allow for including such steps as GTO(j), SBR(j), STO(j), RCL(j) among the imaged functions.

Taking this process a step further, it is not hard to see how any <u>sequence</u> of N program steps can be under full program control. One way would be via a sequence of SBR(i) steps in conjunction with a listing of i values in N consecutive data storage registers, say R_1-R_N. Define F as a function that on its 1st execution inserts the contents of R_1 into I, and on succeeding passes inserts R_2, then R_3, etc. into I. F can readily be implemented using indirect addressability.

Now if each mirror-imaged function has been pre-programmed to implement the desired primary function followed by F, the next SBR(i) step will execute its proper function. We thus have N consecutive reserved program steps available for programming anything we wish by simply specifying a sequence of integers i elsewhere in the program.

That is part of what I had in mind in saying that if manufacturers of hand-held machines proceed in current directions (for example, additional index registers coupled with increased data and program storage capacity), the concept of program self-modification, and with it the calculator/computer differentiation, might become blurred. Indirect addressability is potentially more powerful than many people are aware of.

Another variation, for example, might be to program the 'main sequence,' but precede each of the S steps (or sequences of steps) that might require later program self-modification by a conditional branch to location k. k can be one specific program location where we operate in full generality as already described, plus a list of S consecutive GTO addresses handled by indirect addressability. Alternatively, k may be one of S different locations, if S is small, each with substitute function(s) or provision for generating substitute functions and with a concluding GTO to the proper pre-specified program location.

Either of these methods would allow not only for replacement of any desired function by any other function at any desired location, but for replacement of any consecutive number of program steps by any different number of program steps. This process goes beyond the usual "computer" capability of step-for-step program self-modification.

Thus, given adequate program and data storage capacity (and large capacity was not part of your computer/calculator differentiation), complex "cascading" of indirect addressing is possible so that any number of program steps may be changed in any desired way to be determined by intermediate program results.

The one thing this type of manipulation will not do is to make <u>all</u> program steps available for change. Selectivity and thought must be exercised in the programming process to decide which program steps might need to be altered by the program itself, and in what ways.

But that's part of what programmers do anyway on "self-modifiable" computers. Only certain key routines and locations might need to be changed by the program itself during its operation. Other routines and sequences will obviously remain unmodified throughout program execution.

In other words, lack of full and direct program self-modifiability would simply mean that the programmer must decide ahead of time which program steps and routines might conceivably need modification during program operation and prepare accordingly. But he does this anyway except by different techniques.

Turning now to the question of whether the HP-67 is really an "order of magnitude" advance over the HP-65, I feel I'm on solid ground even taking a 10-fold increase in capability as the criterion.

First, the 224 program steps of the HP-67 actually translates to a 3-fold increase in program space over the 100 step HP-65 when allowance is made for the fully-merged codes of the HP-67--as a detailed multiple-program analysis in the recent 65-Notes demonstrates. Full code merging simply means that any elementary computer operation requires one program step.

For example, STO 9 requires 3 steps on the SR-52 (namely, STO, 00, 09) but only one on the HP-65 or HP-67. The sine function requires 2 steps (f, SIN) on the HP-65, but these steps are 'merged' on the HP-67. STO + 3 (add contents of X register to register 3) takes 3 program steps on the HP-65 but only one step on the HP-25 or HP-67.

You can see how the 224/100 step ratio of the HP-67 to the HP-65 translates into an even greater capability ratio so far as effective program space is concerned. Further, is a 3-fold increase in program space equivalent to only a linear 3-fold increase in power? It's a subjective evaluation, of course, but my programming experience in trying, in Procrustean fashion, to confine deep programs to limited space would suggest that the effective power increase is more like the square of the program space ratio. If so, then there's your 10-fold increase right there.

> Gruenberger to Schwartz at this point: Ah, yes--we are all playing exactly the same game we did around 1953, trying to squeeze a long program into the measly 80 program steps that IBM gave us on the 605, and we'll all look just as silly a few years from now when we have pocket machines with thousands of program steps available.

Next, how much weight shall we place on indirect data register and program register addressability? I've already indicated something of what indirect addressability can accomplish in areas supposedly reserved for a computer with program self-modifiability. The SR-52 sequencer program does what it does only because of indirect program and data register addressing. If someone were to offer me a 224 step HP-65 or a 100 step HP-67 (with no additional features beyond indirect addressing), it would be a difficult choice. This feature, then, may carry about the same weight as the program-size increase.

Now start assigning weights for the following features of the HP-67, and before you're through, I would think a 10-fold increase in capability of the 67 over the 65 might start to seem a conservative estimate.

(1) The HP-67 has 10 user-definable functions (A-E, a-e) vs. 5 for the HP-65, and far greater subroutine depth. On the HP-67, A can call B which can call C which can call D. On the HP-65, if A calls B, B cannot in turn call another user-definable function. Or, if A is called by the main sequence, it can call B which can call C on the HP-67, but on the HP-65 it can call no further. This represents a most significant increase in power and flexibility.

(2) The HP-65 has 2 flags which must be specifically set and reset as desired. The HP-67 not only has 4 flags, but 2 of these will reset automatically when tested--another sgnificant programming tool.

(3) The HP-67 has markedly expanded input capabilities. Not only program steps, but data register contents can be stored on program cards and separately inputted. Besides the flexibility this provides, it leaves program memory free of the step-consuming task of generating needed program constants. In addition, program and data steps may be stored on a single card, manipulated from the keyboard (data/program merge function) and differentiated by a 'smart' card reader.

(4) The HP-67 has markedly expanded output capabilities. There is a 1-second and a 5-second pause display, a stack-review display cycle, and a full register-review display cycle, all programmable. (The program card is readable by the HP-97 which interprets the 5-second pause as a PRINT command.)

(5) The HP-67 has 8 logical-comparison functions:

$x = y$ $\quad x \neq y$ $\quad x \leq y$ $\quad x > y$

$x = 0$ $\quad x \neq 0$ $\quad x > 0$ $\quad x < 0.$

The HP-65 has only the first four, and to perform any of the latter four comparison functions requires additional program steps.

(6) Many other features might be mentioned. There are additional functions such as %, Ch%, D⇋R, and 2-variable statistical functions with retention of sums, sums of ssua squares and sums of products. There is a BST, a line GTO, and a dual function SST (press to display step, release to execute; in RUN mode) for ease of program editing and debugging. DSZ and ISZ are indexible, vs. the single R_8-based DSZ and no ISZ of the HP-65.

There are also 26 registers (i.e., data registers) in the HP-67 compared to 9 for the HP-65. Even those 9 are not fully available since R_9 is shared with the comparison functions and R_8 with DSZ, both of which are frequently needed. In addition, the HP-67 has a P⇋S function which simultaneously exchanges the contents of R_{0-9} with R_{10-19}--a valuable feature.

The only features of the HP-65 not retained in the HP-67 seem to be the OCT-DEC interconversion, and D.MS- (D.MS+ is retained).

The awkwardness, poor code-merging, absence of some important functions (e.g., integer part), and lack of a flexible stack for data manipulation make the SR-52 less superior to the HP-65 that its specs might indicate. The SR-52 sequencer program is a "showcase" program emphasizing its assets and skirting its liabilities, and so is considerably superior to the comparable HP-65 program even with the latter's double Lampman split-logic.

For the average program where indirect addressability is not a major factor, the SR-52 needs far more steps to do what the HP-65 can do, as a comparison of the respective statistical and mathematical program PACs for the two machines will confirm.

I therefore usually program first on the HP-65 for its overall handling ease and, to me, the more direct manner in which mathematical and logical operations are translated into specific program steps without destroying data in the process, as well as the ease with which multiple parameters may be manipulated. I particularly find the SR-52's obligatory branching irritating. When a conditional takes the HP-65 off the main stream, it can insert 2 program steps and continue, or can branch--a most useful option.

The calculator/computer distinction boils down not to unique powers but to comparative ease--and of course there was never any question but that it's easier to operate on instructions as data. You are certainly correct in calling attention to the value of the SR-52's guard digits. In a program with multiple sequential calculations, the number of significant digits can diminish rapidly. Incidentally, the HP-67 has improved algorithms for many functions, and all functions are now accurate to within one count in the 10th digit (vs. one count in the 9th digit for some functions and arguments on earlier HP-65 machines).

The SR-52 has one other unique and valuable feature in comparison to the HP-67. When coupled to its reasonably-priced printer it can provide hard-copy programmed output of generated data as well as program listings, and "tracer" output of a program's step-by-step execution. Once we concede these two points, just about everything else is on the side of the HP-67.

Superficially, the 9 levels of parentheses in the SR-52 might seem preferable to the 4-level stack registers of the HP-67. In program mode, however, the SR-52's parentheses operations allow data to slip away. When the SR-52 performs a logic comparison on two numbers, the compared values are destroyed. When it carries out an

arithmetic or algebraic operation, the original data are again destroyed unless pre-stored elsewhere. In these situations, the HP-67 either keeps the original data in the stack (comparison functions), or automatically preserves the last operand in a "last X" register (functions of one or two variables). In addition, data can be interchanged and manipulated in various ways in the stack itself. You can see how useful these features can be for ease of programming and for conservation of valuable program space.

To some extent, the SR-52 does preserve data in registers 60-69 while building up a parenthsis structure. However, the data is transient and is cleared upon completion of the full parenthsis structure, and is hard to get at and manipulate while in those registers.

In other words, the question of RPN vs. algebraic notation is more than a question of individual preference for ease of use--the usual frame of reference for discussing their relative merits. To a programmer, the more pertinent differential is the striking difference in ability to manipulate, operate upon, compare, and retain data.

At any potential branch point, the SR-52 can choose to branch (GTO or SBR) or to follow the main sequence, depending on the outcome of a comparison test. The HP-67, like the HP-65, has the program-transfer alternative of inserting additional step(s)--2 for the HP-65, 1 for the fully code-merged HP-67--and continuing in the main sequence. Any HP-65 programmer can attest to the value and flexibility of this option.

The hard-wired pre-programmed functions of the HP-67 are more desirable for programming purposes. As you observed, the integer/fraction function is really missed in the SR-52. The programmable 2-variable statistical capability of the HP-67 is most powerful as it includes not just simultaneous cumulative means and standard deviations of X and Y register variables, but retention of the 2-variable sums, sums of squares, and sums of products for further statistical calculation or other program use. Other functions absent from the SR-52 are absolute value, H.MS+, %, Ch%, 3 angular modes, 3 display modes, and rounding.

The HP-67 has a very complete and well thought out system of indirect addressability for data registers and program registers. All 26 data registers including the index register itself may be indirectly accessed for STO and RCL functions and for register arithmetic. Indirect program addressing takes in 20 labels (0-9, A-E, a-e) and all program lines for both GTO and SBR transfers. Subroutine nesting also proceeds to greater depth in the HP-67 (3 levels vs. 2 levels). I feel strongly that as programmers become more familiar with the capabilities of both machines, the HP-67 will be considered a more desirable unit, even allowing for the difference in cost.